

日 本 国 特 許 庁  
JAPAN PATENT OFFICE

17.10.03  
RECEIVED  
04 DEC 2003  
WIPO

別紙添付の書類に記載されている事項は下記の出願書類に記載されている事項と同一であることを証明する。

This is to certify that the annexed is a true copy of the following application as filed with this Office.

出 願 年 月 日                      2 0 0 2 年 1 0 月 1 8 日  
Date of Application:

出 願 番 号                      特 願 2 0 0 2 - 3 0 5 0 1 0  
Application Number:  
[ST. 10/C]:                      [ J P 2 0 0 2 - 3 0 5 0 1 0 ]

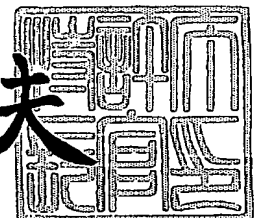
出 願 人                      株式会社アドバンテスト  
Applicant(s):

**PRIORITY DOCUMENT**  
SUBMITTED OR TRANSMITTED IN  
COMPLIANCE WITH  
RULE 17.1(a) OR (b)

2 0 0 3 年 1 1 月 2 0 日

特許庁長官  
Commissioner,  
Japan Patent Office

今 井 康 夫



【書類名】 特許願

【整理番号】 PAVA-14158

【提出日】 平成14年10月18日

【あて先】 特許庁長官殿

【国際特許分類】 G06F 9/45  
G06F 11/28 350  
G06F 11/22 320

【発明者】

【住所又は居所】 東京都練馬区旭町 1 丁目 3 2 番 1 号 株式会社アドバン  
テスト内

【氏名】 前田 裕紀

【特許出願人】

【識別番号】 390005175

【氏名又は名称】 株式会社アドバンテスト

【代理人】

【識別番号】 100089118

【弁理士】

【氏名又は名称】 酒井 宏明

【手数料の表示】

【予納台帳番号】 036711

【納付金額】 21,000円

【提出物件の目録】

【物件名】 明細書 1

【物件名】 図面 1

【物件名】 要約書 1

【プルーフの要否】 要

【書類名】 明細書

【発明の名称】 プログラム開発支援装置、プログラム実行装置、コンパイル方法およびデバッグ方法

【特許請求の範囲】

【請求項 1】 汎用言語ソースプログラムの所定領域に独自仕様言語ソースプログラムが記述された構成の異種言語混在ソースプログラムから所定のプログラム実行装置上で実行可能なプログラムファイルを作成するためのプログラム開発支援装置において、

前記独自仕様言語ソースプログラムをコンパイルして独自仕様言語オブジェクトコードを生成する独自仕様言語コンパイル手段と、

前記異種言語混在ソースプログラム内の前記汎用言語ソースプログラム記述部分をコンパイルして汎用言語オブジェクトコードを生成する汎用言語コンパイル手段と、

前記異種言語混在ソースプログラムから前記独自仕様言語ソースプログラムを抽出し、抽出した独自仕様言語ソースプログラムを指定して前記独自仕様言語コンパイル手段を実行させるとともに、前記異種言語混在ソースプログラムを指定して前記汎用言語コンパイル手段を実行させ、得られた独自仕様言語オブジェクトコードと汎用言語オブジェクトコードを結合してオブジェクトファイルを生成する統合コンパイル手段と、

前記統合コンパイル手段によって生成された少なくとも一つのオブジェクトファイルから前記プログラムファイルを生成するリンク手段と、

を備えたことを特徴とするプログラム開発支援装置。

【請求項 2】 前記統合コンパイル手段は、前記オブジェクトファイルに、前記独自仕様言語オブジェクトコードおよび／または前記汎用言語オブジェクトコードのコード位置情報を付加することを特徴とする請求項 1 に記載のプログラム開発支援装置。

【請求項 3】 前記プログラムファイルを前記プログラム実行装置に転送するプログラム転送手段を備えたことを特徴とする請求項 1 または 2 に記載のプログラム開発支援装置。

【請求項 4】 前記プログラム実行装置に対して該プログラム実行装置に転送されたプログラムファイルを実行させる指示を与えるプログラム実行装置コントロール手段を備えたことを特徴とする請求項 3 に記載のプログラム開発支援装置。

【請求項 5】 前記異種言語混在ソースプログラム内のステートメントにブレークポイントを設定するブレークポイント設定手段と、

前記プログラムファイルの実行時に前記ブレークポイントで該プログラムファイルを停止させるとともに、停止したプログラムファイルの前記ステートメントが前記汎用言語ソースプログラムに属する場合に汎用言語デバッガを起動し、停止したプログラムファイルの前記ステートメントが前記独自仕様言語ソースプログラムに属する場合に独自仕様言語デバッガを起動することを特徴とする請求項 1～4 のいずれか一つに記載のプログラム開発支援装置。

【請求項 6】 前記汎用言語デバッガと前記独自仕様言語デバッガとから得られたデバッグ情報を共通するウィンドウ画面に表示することを特徴とする請求項 5 に記載のプログラム開発支援装置。

【請求項 7】 前記汎用言語は C 言語であり、前記独自仕様言語ソースプログラムは、前記汎用言語ソースプログラムのプリプロセッサコマンドによって該汎用言語ソースプログラム内に記述されたことを特徴とする請求項 1～6 のいずれか一つに記載のプログラム開発支援装置。

【請求項 8】 前記プリプロセッサコマンドは、`#pragma`であることを特徴とする請求項 7 に記載のプログラム開発支援装置。

【請求項 9】 前記プログラム実行装置は、半導体試験装置であることを特徴とする請求項 1～8 のいずれか一つに記載のプログラム開発支援装置。

【請求項 10】 汎用言語ソースプログラムのオブジェクトコードと独自仕様言語ソースプログラムのオブジェクトコードが混在したプログラムファイルを実行するプログラム実行装置において、

前記プログラムファイルの実行開始時に、前記汎用言語ソースプログラムのオブジェクトコードと前記独自仕様言語ソースプログラムのオブジェクトコードをメモリにロードすることを特徴とするプログラム実行装置。

【請求項 1 1】 前記プログラム実行装置は半導体試験装置であることを特徴とする請求項 1 0 に記載のプログラム実行装置。

【請求項 1 2】 汎用言語ソースプログラムの所定領域に独自仕様言語ソースプログラムが記述された構成の異種言語混在ソースプログラムから所定のプログラム実行装置上で実行可能なプログラムファイルを作成するためのコンパイル方法において、

前記異種言語混在ソースプログラムから前記独自仕様言語ソースプログラムを抽出する独自仕様言語ソースプログラム抽出ステップと、

抽出された独自仕様言語ソースプログラムをコンパイルして独自仕様言語オブジェクトコードを生成する独自仕様言語コンパイルステップと、

前記異種言語混在ソースプログラムのうちの前記汎用言語ソースプログラム記述部分をコンパイルして汎用言語オブジェクトコードを生成する汎用言語コンパイルステップと、

前記独自仕様言語オブジェクトコードと前記汎用言語オブジェクトコードを結合してオブジェクトファイルを生成するオブジェクトファイル生成ステップと、

前記オブジェクトファイル生成ステップによって生成された少なくとも一つのオブジェクトファイルから前記プログラムファイルを生成するリンクステップと、

を含んだことを特徴とするコンパイル方法。

【請求項 1 3】 汎用言語ソースプログラムの所定領域に独自仕様言語ソースプログラムが記述された構成の異種言語混在ソースプログラムから作成された所定のプログラム実行装置上で実行可能なプログラムファイルをデバッグするためのデバッグ方法において、

前記異種言語混在ソースプログラム内のステートメントにブレークポイントを設定するブレークポイント設定ステップと、

前記プログラムファイルの実行時に前記ブレークポイントで該プログラムファイルを停止させるとともに、停止したプログラムファイルの前記ステートメントが前記汎用言語ソースプログラムに属する場合に汎用言語デバッガを起動し、停止したプログラムファイルの前記ステートメントが前記独自仕様言語ソースプロ

グラムに属する場合に独自仕様言語デバッガを起動するデバッガ起動ステップと、

前記汎用言語デバッガと前記独自仕様言語デバッガとから得られたデバッグ情報を共通するウィンドウ画面に表示するデバッグ情報表示ステップと、  
を含んだことを特徴とするデバッグ方法。

#### 【発明の詳細な説明】

##### 【0001】

##### 【発明の属する技術分野】

本発明は、半導体試験装置等の被制御装置に対する制御コマンドが記述された独自仕様言語プログラムと、独自仕様言語プログラムの実行ステップや独自仕様言語プログラムから得られたデータの処理ステップ等が記述された汎用言語プログラムとを開発するためのプログラム開発支援装置、それらプログラムを実行するプログラム実行装置、それらプログラムのコンパイル方法およびデバッグ方法に関し、特に、上記した独自仕様言語プログラムと汎用言語プログラムとが一つのファイル内に混在して記述された異種言語混在プログラムを開発するためのプログラム開発支援装置、プログラム実行装置、コンパイル方法およびデバッグ方法に関する。

##### 【0002】

##### 【従来の技術】

膨大かつ多様な信号処理が求められる測定装置や通信装置などの電子機器の多くには、高性能なプロセッサが搭載されている。それに伴い、プロセッサ上で実行されるプログラム（ファームウェア）もまた、複雑でサイズが大きくなる傾向にある。よって、必然と、これら電子機器に記録されているプログラムは、未知のバグを含んでいたり、機能の追加や改善を求められることが多くなっている。

##### 【0003】

そのため、このような電子機器は、動作の設定／監視や上記したプログラムのアップデートを可能にするために、通信ケーブルを介して、外部のコンピュータと接続可能な形態をとっていることが多い。すなわち、この形態では、外部のコンピュータから電子機器のプロセッサに対し、制御コマンドや制御プログラム自

体を配信することが可能になっている。さらに換言すれば、外部のコンピュータ上で開発したプログラムのアルゴリズムや設定内容に従って、電子機器を操作することが可能になっている。

#### 【0004】

そのような電子機器のうちでも特に半導体試験装置は、多種多様かつ独自仕様の半導体デバイスに対してそれぞれ個別のデバイステストプログラムを用意する必要がある特殊な測定装置であり、半導体試験装置のプロセッサ上で実行させるプログラムを、利用者（半導体デバイス製造メーカ）自らが開発するという形態が定着している。

#### 【0005】

ところが、半導体試験装置を筆頭とした特殊用途の電子機器は、搭載されるプロセッサも独自仕様のものが多く、必然とそのプロセッサが理解できるバイナリファイルも独自仕様に従ったものとなっている。そのため、そのようなバイナリファイルの元となるソースファイルの作成に必要なプログラム言語の仕様や開発支援環境もまた特殊となり、プログラム開発者は、プログラム言語とともに開発支援環境の操作も習熟する必要があった。

#### 【0006】

このような背景から、近年においては、C言語やJ A V A（登録商標）等の汎用言語によって開発されたプログラムを実行することができる電子機器も登場してきているが、そのような電子機器を採用することは、過去の独自仕様のプログラム資産を活用することができないという新たな問題を生み出す。

#### 【0007】

そこで、現状では、データ処理やアルゴリズム全体を、C言語のような汎用言語プログラムで記述し、その汎用言語プログラム内から、電子機器に依存する独自仕様言語プログラムをサブルーチンとして呼び出すという形態がみられる。以下に、この形態におけるプログラムの開発と実行に関し、プログラムを実行する電子機器が半導体試験装置である場合を例に挙げて詳述する。

#### 【0008】

まず、半導体試験装置上でのプログラムの実行動作を理解するために、半導体

試験装置とそのプログラム開発環境について説明する。半導体試験装置は、半導体メモリ、ロジック IC、リニア IC などの半導体デバイスに対して所定の動作試験を行う特殊な測定装置であり、一般にその装置構成は、上記した半導体デバイスの種別ごとに異なっている。また、半導体試験装置への試験実行指示、試験結果取得およびプログラム開発は、通常、半導体試験装置に接続されたワークステーションによって行なわれ、半導体試験は、半導体試験装置とワークステーションとで構成される半導体試験システム（例えば、特許文献 1 参照）によって実現される。

#### 【0009】

図 10 は、従来の半導体試験システムの概略構成を示すブロック図であり、特に、上記したように異なる装置構成の半導体試験装置間において共通する構成を示すものである。図 10 において、半導体試験装置 100 は、テストプロセッサ 110 と、テスト本体 120 と、テストヘッド 130 と、通信インターフェース 140 とを備えて構成されている。

#### 【0010】

テストプロセッサ 110 は、テスト本体 120 との間で、制御コマンドの送信や試験データの送受信を行う手段であり、テスト本体 120 の制御や後述するワークステーションとの間の通信を行うコントローラである。特に、テストプロセッサ 110 は、内部に搭載したメモリ（図示略）に、OS（オペレーティングシステム）カーネル 111 を格納しており、デバイステストプログラムの起動や監視、メモリ管理などを行うとともに、同じくメモリに格納された通信バスドライバ 112 やテストバスドライバ 113 を介して、通信インターフェース 140 の監視／制御、テスト本体 120 の制御、試験データの送受信を行う。

#### 【0011】

ここで、デバイステストプログラムは、上記した汎用言語プログラム 114 と独自仕様言語プログラム 117 とで構成されており、それら全体で、被測定デバイス 131 に対する機能試験や DC パラメトリック試験等の各種の試験を実施する手順等を規定している。汎用言語プログラム 114 は、試験結果として取得した各種データを処理するコマンドを含んだステートメントと、デバイステストプ



プログラム全体をどのように実行するかを示すコマンドを含んだステートメントとによって構成されており、OSカーネル111上で直接実行可能なバイナリファイルである。

#### 【0012】

一方、独自仕様言語プログラム117は、テスト本体120を制御するためのコマンドで構成されるオブジェクトファイルである。但し、そのオブジェクトファイルは、過去の資産である独自仕様言語プログラムと同様に、独自仕様言語プログラム117に最適化されたカーネル上でのみ直接実行可能なバイナリファイルである。そのため、独自仕様言語プログラム117をOSカーネル111上で実行させるにあたっては、実行用エミュレータ115による解釈処理を要する。また、独自仕様言語プログラム117は、後述するワークステーション200に対してのディスク・アクセス、キー入力、ディスプレイ表示といった入出力コマンドも含んでおり、それら入出力コマンドの実行にあたっては、実行用エミュレータ115による解釈に加え、さらにIO制御用エミュレータ116による解釈処理を要する。

#### 【0013】

テスト本体120は、テストプロセッサ110から送信される制御コマンドに従って、テストヘッド130に実装された被測定デバイス131に対し、機能試験やDCパラメトリック試験、RF試験（高周波試験）等の各種の試験を行う手段であり、レジスタ121、メモリ122、試験信号送受信部123を備えて構成されている。レジスタ121は、テストプロセッサ110内の上記テストバスドライバ113との間で送受信される各種のデータを格納し、格納されたデータは、直接またはメモリ122を介して試験信号送受信部123に渡される。

#### 【0014】

また、試験信号送受信部123から出力されるデータは、一旦レジスタ121やメモリ122に格納された後、レジスタ121を介してテストプロセッサ110内の上記テストバスドライバ113に渡される。試験信号送受信部123は、パターン発生器やタイミング発生器、DCユニット等の種々の試験ユニットから構成され、これら試験ユニットによって生成された試験信号を被測定デバイス1

31に入力するとともに、被測定デバイス131の出力ピンに現れるデータを取得する。

#### 【0015】

図11は、上記したワークステーション200の概略構成を示すブロック図である。ワークステーション200は、半導体試験装置100内のテストプロセッサ110に対してプログラムの転送や実行指示を行うコンソール端末の役割と、上記した汎用言語プログラム114や独自仕様言語プログラム117の開発を支援するプログラム開発支援装置の役割とを担う。図11において、ワークステーション200は、プロセッサ220と、通信インターフェース241と、ハードディスク装置242と、マウス243と、キーボード244と、ディスプレイ装置245とを備えて構成されている。

#### 【0016】

プロセッサ220は、内部に搭載したメモリ（図示略）に、OS（オペレーティングシステム）カーネル221を格納しており、種々のプログラムの起動や監視、メモリ管理などを行うとともに、同じくメモリに格納される通信バスドライバ223、ハードディスクドライバ224、マウスドライバ225、キーボードドライバ226、ディスプレイドライバ227を介して、通信インターフェース241の監視や制御、ハードディスク装置242との間でのプログラムやデータの読み込み／書き込み、マウス243やキーボード244からの入力情報の取得、ディスプレイ装置245への表示情報の出力などを行う。ここで特に、通信インターフェース241は、通信ケーブル（図示せず）を介して、図10に示した通信インターフェース140と接続されており、ワークステーション200と半導体試験装置100との間の通信を可能としている。

#### 【0017】

また、OSカーネル221は、GUI（Graphical User Interface）処理部222を有しており、図示するエディタ228、汎用言語コンパイラ229、リンカ233、汎用言語デバッガ231、独自仕様言語コンパイラ230、独自仕様言語デバッガ232などの各種プログラムは、ディスプレイ装置245上に表示されるウィンドウ画面単位で実行可能である。なお、

このワークステーション 200 は、汎用的なコンピュータの装置構成と同等である。よって、上記した各種ドライバや各種プログラムは、通常はハードディスク装置 242 に記憶されており、OS カーネル 221 に従って必要に応じて上記メモリに読み込まれて実行される。

#### 【0018】

次に、上記した半導体試験装置 100 とワークステーション 200 とで構成される半導体試験システムにおいて、デバイステストプログラムの開発と実行の流れについて説明する。図 12 は、従来のデバイステストプログラムの開発と実行手順を示したフローチャートである。ここでは、デバイステストプログラムが、上記したように汎用言語プログラムと独自仕様言語プログラムとで構成されており、汎用言語プログラムとして C 言語を採用し、独自仕様言語プログラムとして ATL（アドバンテスト社独自規格）を採用した場合を例に挙げる。

#### 【0019】

まず、プログラム開発者は、ワークステーション 200 上においてエディタ 228 を起動させ、C 言語によるソースプログラムを作成する（ステップ S301）。このソースプログラムは、上述したように、デバイステストプログラム全体のアルゴリズムを記述するものであり、シーケンス処理の所望の位置で、ATL で記述されたオブジェクトプログラムを呼び出して実行したり、その実行によって得られた試験結果データを処理する手順を規定している。

#### 【0020】

C 言語によるソースプログラムの作成を終えると、プログラム開発者は、C コンパイラ（汎用言語コンパイラ 229 に相当）に対し、作成したソースプログラムのファイル（以下、必要なヘッダファイル等を含めて C ソースファイルと称する。）を指定してコンパイルを実行させる（ステップ S302）。コンパイル処理では、まず構文チェックが行なわれ、構文エラーが生じた場合には、エディタ 228 でエラー箇所を修正し、再度コンパイルを実行する。エラーがない場合には、上記した C ソースファイルを機械語に翻訳したオブジェクトファイル（以下、C オブジェクトファイルと称する。）が生成される。

#### 【0021】

ステップS301で作成された複数のCソースファイルに対し、上記したステップS302が完了すると、プログラム開発者は、リンカ233に対し、生成された複数のCオブジェクトファイルやその他に必要なライブラリファイルを指定してリンクを実行させる（ステップS303）。このリンクによって、半導体試験装置100のテストプロセッサ110上で直接実行可能な単一のCオブジェクトファイルが生成される。

#### 【0022】

また、プログラム開発者は、C言語によるオブジェクトファイルの生成と並行して、ワークステーション200上においてエディタ228を起動させ、ATLによるソースプログラムを作成する（ステップS401）。このソースプログラムは、上述したように、半導体試験装置100を制御するための制御コマンドを記述するものである。

#### 【0023】

ATLによるソースプログラムの作成が終わると、プログラム開発者は、ATLコンパイラ（独自仕様言語コンパイラ230に相当）に対し、作成したソースプログラムのファイル（以下、ATLソースファイルと称する。）を指定してコンパイルを実行させる（ステップS402）。なお、このコンパイル処理でも、上記したステップS302と同様に、まず構文チェックが行なわれ、構文エラーが生じた場合には、エディタ228でエラー箇所を修正し、再度コンパイルを実行する。エラーがない場合には、上記したATLソースプログラムは、上記したCオブジェクトファイルで表わされる機械語（ある特定のテストプロセッサで理解できる機械語）とは異なる旧テストプロセッサ仕様の機械語に翻訳され、オブジェクトファイル（以下、ATLオブジェクトファイルと称する。）が生成される。

#### 【0024】

このような手順によって、単一CオブジェクトファイルとATLオブジェクトファイル群が用意されると、プログラム開発者は、ワークステーション200上において、半導体試験装置100との通信を可能にするコントロールプログラムを起動し、そのコントロールプログラムを用いて単一Cオブジェクトファイルと

A T Lオブジェクトファイル群を半導体試験装置100のテストプロセッサ110へと転送する（ステップS304，ステップS403）。

#### 【0025】

続いて、プログラム開発者は、上記したコントロールプログラムに対して、単一Cオブジェクトファイルの実行指示を与える（ステップS305）。これにより、半導体試験装置100のテストプロセッサ110は、単一Cオブジェクトファイルに記述されたアルゴリズムに従って、A T Lオブジェクトファイルの実行→テスト本体120の所望の試験ユニットの稼動→被測定デバイス131から得られた試験結果の取得→データ処理を繰り返す。この際、データ処理によって適宜加工等が行われた試験結果は、半導体試験装置100の通信インターフェース140、通信ケーブル、ワークステーション200の通信インターフェース241を介して、上記したコントロールプログラムで受け取ることができ、そのコントロールプログラムに割り当てられたウィンドウ画面上に表示される。

#### 【0026】

ここで、プログラム開発者は、試験結果が明らかに異常である場合等の不具合を発見した場合、デバイステストプログラムに論理的なエラーが含まれていると判断し、ワークステーション200上において汎用言語デバッガ231を起動させ、Cソースファイル中の所定のステートメントにブレークポイントを設定する。そして、プログラム開発者がデバッグ開始を指示することにより、汎用言語デバッガ231は、再度、上記したステップS302～S305の手順によって単一Cオブジェクトファイルを実行し、実行されたステートメントが、設定したブレークポイントに達したことを検出すると、ブレークしたステートメントの段階で有効となっている変数を表示する。プログラム開発者は、この変数の確認によって論理的なエラーを発見すると、エディタ228を起動させ、Cソースファイルを適宜修正して上記したステップS302～S305の手順を繰り返す。

#### 【0027】

一方、プログラム開発者は、汎用言語デバッガ231によってCソースファイルの論理的なエラーを発見できないときは、続いて、独自仕様言語デバッガ232を起動させ、A T Lソースファイル中の所定のステートメントにブレークポイ

ントを設定し、上記同様のデバッグ処理を行う。

【0028】

【特許文献1】

特開 2001-195275号公報

【0029】

【発明が解決しようとする課題】

しかしながら、上記したように、被制御装置（上記例では半導体試験装置）上で実行させるプログラムを汎用言語と独自仕様言語のように異種言語で記述した場合には、独自仕様言語による過去のプログラム資産を活用できるものの、双方のプログラムの開発段階において、それぞれ個別のソースファイルを必要とする。上記した例では、CソースファイルとATLソースファイルとをそれぞれ別ファイルとして用意する必要がある。簡単に言えば、汎用言語で記述されたソースファイル中に独自仕様言語に作成されたオブジェクトファイルの呼び出しを埋め込んだ場合には、少なくとも2つのソースファイルが必要となる。特に、ある一つの汎用言語ソースファイルに対しては、ある特定の独自仕様言語ソースファイルが対応するため、これらファイルはひとまとめで管理しなければならない。

【0030】

さらに、双方のソースファイルの内容から、関連する異種言語のソースファイルを特定することは困難であり、一度管理を誤ると、対応関係を再度見出すのに多くの時間と手間を要するという問題があった。このことから、エディタ上でのソースファイルの修正作業や他のソースファイルの部分的な流用については慎重にならざるを得ず、プログラム開発者にとってもミスを増やす原因となっていた。

【0031】

また、一つの実行プログラムに対して、汎用言語ソースファイルと独自仕様言語ソースファイルとを用意することから、必然と、それらをコンパイルすることによって同じ数だけオブジェクトファイルが生成される。これは、上記した管理がさらに複雑になることを意味する。このように、従来では、一つの実行プログラムに対して異種言語の複数のソースファイルおよびオブジェクトファイルが存

在したため、ファイル管理が煩雑となり、プログラムの開発効率を低下させるという問題があった。

#### 【0032】

本発明は上記に鑑みてなされたものであって、汎用言語ソースファイルのプリプロセッサ記述部に独自仕様言語のソースファイルを埋め込むことによって、独自仕様言語のプログラムによる過去の資産を活用できるとともに、一つの実行ファイルに対して必要なソースファイルおよびオブジェクトファイルの数を大幅に減少させることのできるプログラム開発支援装置、プログラム実行装置、コンパイル方法およびデバッグ方法を提供することを目的とする。

#### 【0033】

##### 【課題を解決するための手段】

上記目的を達成するために、請求項1にかかるプログラム開発支援装置は、汎用言語ソースプログラムの所定領域に独自仕様言語ソースプログラムが記述された構成の異種言語混在ソースプログラムから所定のプログラム実行装置上で実行可能なプログラムファイルを作成するためのプログラム開発支援装置において、前記独自仕様言語ソースプログラムをコンパイルして独自仕様言語オブジェクトコードを生成する独自仕様言語コンパイル手段（後述する独自仕様言語コンパイラ30に相当）と、前記異種言語混在ソースプログラム内の前記汎用言語ソースプログラム記述部分をコンパイルして汎用言語オブジェクトコードを生成する汎用言語コンパイル手段（後述する汎用言語コンパイラ29に相当）と、前記異種言語混在ソースプログラムから前記独自仕様言語ソースプログラムを抽出し、抽出した独自仕様言語ソースプログラムを指定して前記独自仕様言語コンパイル手段を実行させるとともに、前記異種言語混在ソースプログラムを指定して前記汎用言語コンパイル手段を実行させ、得られた独自仕様言語オブジェクトコードと汎用言語オブジェクトコードを結合してオブジェクトファイルを生成する統合コンパイル手段（後述する統合コンパイラ34に相当）と、前記統合コンパイル手段によって生成された少なくとも一つのオブジェクトファイルから前記プログラムファイルを生成するリンク手段（後述するリンカ33に相当）と、を備えたことを特徴としている。

**【0034】**

この発明によれば、汎用言語プログラムで記述されるアルゴリズム内に所定のプログラム実行装置特有の独自仕様言語プログラムの命令を組み込んだ態様において、独自仕様言語プログラムのソース自体が汎用言語プログラムのソース内に埋め込まれているので、独自仕様言語プログラムと汎用言語プログラムを、ソースファイルのみでなく、コンパイルによって作成されるオブジェクトファイルの段階においても一つのファイルとして取り扱うことができる。

**【0035】**

また、請求項2にかかるプログラム開発支援装置は、上記した発明において、前記統合コンパイル手段が、前記オブジェクトファイルに、前記独自仕様言語オブジェクトコードおよび／または前記汎用言語オブジェクトコードのコード位置情報を付加することを特徴としている。

**【0036】**

この発明によれば、コード位置情報を参照して、オブジェクトファイルから、独自仕様言語オブジェクトコードと汎用言語オブジェクトコードを取り出すことができる。

**【0037】**

また、請求項3にかかるプログラム開発支援装置は、上記した発明において、前記プログラムファイルを前記プログラム実行装置に転送するプログラム転送手段（後述するコントロールプログラムに相当する）を備えたことを特徴としている。

**【0038】**

この発明によれば、所定のプログラム実行装置へのファームウェアアップデートを行うことができる。

**【0039】**

また、請求項4にかかるプログラム開発支援装置は、上記した発明において、前記プログラム実行装置に対して該プログラム実行装置に転送されたプログラムファイルを実行させる指示を与えるプログラム実行装置コントロール手段（後述するコントロールプログラムに相当）を備えたことを特徴としている。



## 【 0 0 4 0 】

この発明によれば、所定のプログラム実行装置へのプログラム実行指示、デバッグ等を行うことができる。

## 【 0 0 4 1 】

また、請求項 5 にかかるプログラム開発支援装置は、上記した発明において、前記異種言語混在ソースプログラム内のステートメントにブレークポイントを設定するブレークポイント設定手段（後述する統合デバッガ 3 5 に相当）と、前記プログラムファイルの実行時に前記ブレークポイントで該プログラムファイルを停止させるとともに、停止したプログラムファイルの前記ステートメントが前記汎用言語ソースプログラムに属する場合に汎用言語デバッガ（後述する汎用言語デバッガ 3 1 に相当）を起動し、停止したプログラムファイルの前記ステートメントが前記独自仕様言語ソースプログラムに属する場合に独自仕様言語デバッガ（後述する独自仕様言語デバッガ 3 2 に相当）を起動することを特徴としている。

## 【 0 0 4 2 】

この発明によれば、独自仕様言語ソースプログラムとの汎用言語ソースプログラムとが混在した異種言語混在プログラムに対してデバッグを行う場合にも、既存するそれぞれのデバッガを利用することができる。

## 【 0 0 4 3 】

また、請求項 6 にかかるプログラム開発支援装置は、上記した発明において、前記汎用言語デバッガと前記独自仕様言語デバッガとから得られたデバッグ情報を共通するウィンドウ画面に表示することを特徴としている。

## 【 0 0 4 4 】

この発明によれば、独自仕様言語ソースプログラムとの汎用言語ソースプログラムとが混在した異種言語混在プログラムに対してデバッグを行う場合にも、共通のデバッグ環境を利用することができる。

## 【 0 0 4 5 】

また、請求項 7 にかかるプログラム開発支援装置は、上記した発明において、前記汎用言語は C 言語であり、前記独自仕様言語ソースプログラムは、前記汎用

言語ソースプログラムのプリプロセッサコマンドによって該汎用言語ソースプログラム内に記述されたことを特徴としている。

【0046】

また、請求項8にかかるプログラム開発支援装置は、上記した発明において、前記プリプロセッサコマンドは、`#pragma`であることを特徴としている。

【0047】

また、請求項9にかかるプログラム開発支援装置は、上記した発明において、前記プログラム実行装置は、半導体試験装置であることを特徴としている。

【0048】

また、請求項10にかかるプログラム実行装置は、汎用言語ソースプログラムのオブジェクトコードと独自仕様言語ソースプログラムのオブジェクトコードが混在したプログラムファイルを実行するプログラム実行装置（後述する半導体試験装置11に相当）において、前記プログラムファイルの実行開始時に、前記汎用言語ソースプログラムのオブジェクトコードと前記独自仕様言語ソースプログラムのオブジェクトコードをメモリにロードすることを特徴としている。

【0049】

この発明によれば、独自仕様言語ソースプログラムとの汎用言語ソースプログラムとが混在した異種言語混在プログラムを実行することができる。

【0050】

また、請求項11にかかるプログラム実行装置は、上記した発明において、該プログラム実行装置が半導体試験装置であることを特徴としている。

【0051】

また、請求項12にかかるコンパイル方法は、汎用言語ソースプログラムの所定領域に独自仕様言語ソースプログラムが記述された構成の異種言語混在ソースプログラムから所定のプログラム実行装置上で実行可能なプログラムファイルを作成するためのコンパイル方法において、前記異種言語混在ソースプログラムから前記独自仕様言語ソースプログラムを抽出する独自仕様言語ソースプログラム抽出ステップ（後述するステップS121に相当）と、抽出された独自仕様言語ソースプログラムをコンパイルして独自仕様言語オブジェクトコードを生成する

独自仕様言語コンパイルステップ（後述するステップ S 1 2 3 に相当）と、前記異種言語混在ソースプログラムのうちの前記汎用言語ソースプログラム記述部分をコンパイルして汎用言語オブジェクトコードを生成する汎用言語コンパイルステップ（後述するステップ S 1 2 2 に相当）と、前記独自仕様言語オブジェクトコードと前記汎用言語オブジェクトコードを結合してオブジェクトファイルを生成するオブジェクトファイル生成ステップ（後述するステップ S 1 2 4 に相当）と、前記オブジェクトファイル生成ステップによって生成された少なくとも一つのオブジェクトファイルから前記プログラムファイルを生成するリンクステップ（後述するステップ S 1 3 0 に相当）と、を含んだことを特徴としている。

#### 【0052】

また、請求項 13 にかかるデバッグ方法は、汎用言語ソースプログラムの所定領域に独自仕様言語ソースプログラムが記述された構成の異種言語混在ソースプログラムから作成された所定のプログラム実行装置上で実行可能なプログラムファイルをデバッグするためのデバッグ方法において、前記異種言語混在ソースプログラム内のステートメントにブレークポイントを設定するブレークポイント設定ステップと、前記プログラムファイルの実行時に前記ブレークポイントで該プログラムファイルを停止させるとともに、停止したプログラムファイルの前記ステートメントが前記汎用言語ソースプログラムに属する場合に汎用言語デバッガを起動し（後述するステップ S 2 0 3 に相当）、停止したプログラムファイルの前記ステートメントが前記独自仕様言語ソースプログラムに属する場合に独自仕様言語デバッガを起動する（後述するステップ S 2 0 6 に相当）デバッガ起動ステップと、前記汎用言語デバッガと前記独自仕様言語デバッガとから得られたデバッグ情報（後述するステップ S 2 0 4，ステップ S 2 0 7 に相当）を共通するウィンドウ画面に表示するデバッグ情報表示ステップ（後述するステップ S 2 0 5 に相当）と、を含んだことを特徴としている。

#### 【0053】

##### 【発明の実施の形態】

以下に、本発明にかかるプログラム開発支援装置、プログラム実行装置、コンパイル方法およびデバッグ方法の実施の形態を図面に基づいて詳細に説明する。

なお、この実施の形態により本発明が限定されるものではない。

#### 【0054】

この実施の形態では、本発明の特徴の理解を容易にするために、上述した従来技術の説明と同様に、本発明を、半導体試験装置とワークステーションとから構成される半導体試験システムに適用した場合について説明する。具体的には、本発明にかかるプログラム開発支援装置が、半導体試験システムのワークステーションに相当し、本発明にかかるプログラム実行装置が半導体試験システムの半導体試験装置に相当し、本発明にかかるコンパイル方法およびデバッグ方法が上記半導体試験システム上でのコンパイル方法およびデバッグ方法に相当する。

#### 【0055】

図1は、本実施の形態にかかる半導体試験システムを示すブロック図である。図1に示す半導体試験システムは、通信ケーブルで接続されたワークステーション10と半導体試験装置11とを備えて構成される。

#### 【0056】

ワークステーション10の基本構成は、図11に示した従来の半導体試験システムのワークステーション200と変わることはなく、図1中のプロセッサ20、通信インターフェース41、ハードディスク装置42と、マウス43と、キーボード44と、ディスプレイ装置45は、それぞれ図11に示したプロセッサ220、通信インターフェース241、ハードディスク装置242、マウス243、キーボード244、ディスプレイ装置245に対応する。

#### 【0057】

また、プロセッサ20内部において、メモリ（図示略）に格納されるOSカーネル21、GUI処理部22、通信バスドライバ23、ハードディスクドライバ24、マウスドライバ25、キーボードドライバ26、ディスプレイドライバ27、エディタ28、汎用言語コンパイラ29、独自仕様言語コンパイラ30、汎用言語デバッガ31、独自仕様言語デバッガ32、リンカ33もまた、それぞれ図11に示したOSカーネル221、GUI処理部222、通信バスドライバ223、ハードディスクドライバ224、マウスドライバ225、キーボードドライバ226、ディスプレイドライバ227、エディタ228、汎用言語コンパイ

ラ 229、独自仕様言語コンパイラ 230、汎用言語デバッガ 231、独自仕様言語デバッガ 232、リンカ 233と同様に機能する。

#### 【0058】

ここで、本実施の形態で説明するワークステーション 10において、従来のワークステーション 200と異なるのは、上記した汎用言語コンパイラ 29と独自仕様言語コンパイラ 30の上位アプリケーションに位置する統合コンパイラ 34を有している点である。換言すれば、このワークステーション 10は、統合コンパイラ 34から、汎用言語コンパイラ 29と独自仕様言語コンパイラ 30をそれぞれ利用できることを特徴としている。

#### 【0059】

他にも、本実施の形態で説明するワークステーション 10では、従来のワークステーション 200と異なる点として、上記した汎用言語デバッガ 31と独自仕様言語デバッガ 32の上位アプリケーションに位置する統合デバッガ 35を有している。すなわち、上記した統合コンパイラ 34と同様に、このワークステーション 10は、統合デバッガ 35から、汎用言語デバッガ 31と独自仕様言語デバッガ 32をそれぞれ利用できることを特徴としている。

#### 【0060】

また、図 1 に示す半導体試験システムでは、ワークステーション 10に半導体試験装置 11が接続されているが、この半導体試験装置 11の内部構成については図 10 に示した従来の半導体試験装置 100と同様であるので、ここではその説明を省略する。但し、この半導体試験装置 11は、ワークステーション 10において開発されるプログラムの態様によっては、テストプロセッサの動作が従来とは一部異なる。その説明については後述する。

#### 【0061】

次に、この半導体試験システムにおいて、デバイステストプログラムの開発と実行の流れについて説明する。図 2 は、本実施の形態におけるデバイステストプログラムの開発と実行手順を示したフローチャートである。ここでは、図 12 の説明と同様に、デバイステストプログラムが、汎用言語プログラムと独自仕様言語プログラムとで構成されており、汎用言語プログラムとして C 言語を採用し、

独自仕様言語プログラムとしてATL（アドバンテスト社独自規格）を採用した場合を例に挙げる。

#### 【0062】

まず、プログラム開発者は、ワークステーション10上においてエディタ28を起動させ、ソースプログラムを作成する（ステップS110）。ここで、このソースプログラムは、汎用言語であるC言語で記述されるが、図12で説明したCソースファイルの内容とは異なり、プリプロセッサ記述部に、独自仕様言語であるATLソースファイルの内容を記述することを特徴としている。ここで、このソースプログラムをC+ATLソースプログラムと称する。

#### 【0063】

図3は、C+ATLソースプログラムの記述例を示す図である。なお、図3において、左端に配置された「数字：」は行番号を示しており、ここでは説明の便宜上用いるが、実際のプログラム動作においては無視される。以下のC+ATLソースプログラムの内容の説明では、この行番号によって各ステートメントを参照する。

#### 【0064】

C言語コンパイラは、先頭が#で続くステートメントをプリプロセッサコマンドであると認識する。図3に示すC+ATLソースプログラムでは、行番号1の`#include`、行番号3, 4, 5の`#pragma`がそのプリプロセッサコマンドに相当する。`#include`は、ヘッダファイル「AT/hybrid.h」の記述内容を単にその位置に展開するのみであり、行番号10以降に続くメイン関数内において必要となる内容である。

#### 【0065】

一方の`#pragma`は、C言語との全般的な互換性を保ちながら、マシンまたはOSに固有の機能を実現する特別なプリプロセッサコマンドである。よって、`#pragma`は定義上、マシンまたはOS固有であり、通常コンパイラごとに異なる。`#pragma`は、通常、プリプロセッサに新しい機能を与えたり、インプリメントに依存する情報をコンパイラに伝えるといった利用方法が主であるが、本実施の形態で作成されるC+ATLソースプログラムでは、`#prag`

maによって渡されるトークンとして、独自仕様言語であるATLの記述を埋め込むことを特徴としている。なお、この`#pragma`によって渡されるATLの記述の処理については後述する。

#### 【0066】

図3に示すC+ATLソースプログラムにおいて、行番号10～28に記述されている内容は、従来の半導体試験システムのワークステーション上において作成される内容と同じものであり、その記述の中において、ATLオブジェクトファイルの呼び出しやデータ処理が規定されている。

#### 【0067】

C+ATLソースプログラムの作成が終えると、プログラム開発者は、統合コンパイラ34に対し、作成したC+ATLソースプログラムのファイル（以下、必要なヘッダファイル等を含めてC+ATLソースファイルと称する。）を指定してコンパイルを実行させる（ステップS120）。このコンパイルの実行に際しては、まず構文チェックが行われ、構文エラーが生じた場合には、エディタ28でエラー箇所を修正し、再度コンパイルを実行する。エラーがない場合に初めて、オブジェクトファイルを生成するためのコンパイル処理が開始される。

#### 【0068】

図4は、統合コンパイラ34によるコンパイル処理を説明するためのフローチャートである。統合コンパイラ34は、ステップS110において作成されたC+ATLソースファイルにおいて構文エラーを発見できなかったときは、続いて、そのC+ATLソースファイルからATL記述部を抽出し、ATLソースファイルを生成する（ステップS121）。図5は、このATLソースファイルの生成処理を説明するための説明図である。ATLソースファイルの生成処理は、上述したように、C+ATLソースファイルのプリプロセッサ記述部から、`#pragma`を識別し、`#pragma`の後に続くトークンを解析することから始まる。図5に示す例では、`#pragma`の直後の`atl`が、その後に続く情報がATL記述部であることを示すキーワードとなる。

#### 【0069】

図5の例を用いて具体的な説明を行うと、統合コンパイラ34は、行番号3に

において、`#pragma atl`を認識すると、その後に続くキーワードの`name`を取り出し、そのキーワード`name`に続くダブルクォーテーションで囲まれた記述、すなわち`SAMPLE`がプログラム名であると解釈する。この解釈によって、`ATL`ソースファイルの先頭部に、`PRO SAMPLE`が挿入される。さらに、行番号4において、`#pragma atl`を認識すると、その後に続くキーワードの`socket`を取り出し、そのキーワード`socket`に続くダブルクォーテーションで囲まれた記述、すなわち`SSOC`が使用する`ATL`のソケットプログラム名であると解釈する。この解釈によって、`ATL`ソースファイル内の`PRO SAMPLE`の後に`SSOC`が挿入される。

#### 【0070】

さらに、統合コンパイラ34は、行番号5において、`#pragma atl`を認識すると、その後に続くキーワードの`proc`を取り出し、そのキーワード`proc`の直後の文字列とその後に続くダブルクォーテーションで囲まれた記述、すなわち、

```
P1 (ARG1, ARG2 (2)) " {WRITE " ARG1=" , ARG1, /WRITE " ARG2=" , ARG2, /} "
```

が関数定義であると解釈する。この解釈によって、`ATL`ソースファイルに、

```
P1: ARGUMENT (ARG1, ARG2 (2))
    WRITE " ARG1=" , ARG1, /
    WRITE " ARG2=" , ARG2, /
    GOTO CONTINUE
```

が追記される。

#### 【0071】

そして、統合コンパイラ34は、`C+ATL`ソースファイルにおいて、`#pragma atl`を発見することができないまま、`C+ATL`ソースファイルの最終行（行番号28）に達すると、`ATL`ソースファイルの最後に`END`を挿入してその`ATL`ソースファイルの作成を終える。

#### 【0072】

`ATL`ソースファイルの作成が終わると、統合コンパイラ34は、`C+ATL`



ソースファイルのC言語記述部のコンパイル、すなわちCオブジェクトコードの生成を行う（ステップS 1 2 2）。このコンパイルは、通常のCコンパイラ（汎用言語コンパイラ 2 9 に相当する。）によって処理され、上記した# p r a g m a の記述部分は無視される。なお、ここでは、統合コンパイラ 3 4 がCコンパイラを呼び出して処理させるとしているが、統合コンパイラ 3 4 自体にCコンパイラの機能を取り入れ、上記したA T Lソースファイルの生成と並行して、Cオブジェクトコードの生成を行ってもよい。この場合、図 1 に示した汎用言語コンパイラ 2 9 は不要となる。

#### 【 0 0 7 3 】

Cオブジェクトコードの生成が終わると、統合コンパイラ 3 4 は、ステップS 1 2 1において生成されたA T Lソースファイルのコンパイル、すなわちA T Lオブジェクトコードの生成を行う（ステップS 1 2 3）。このコンパイルは、A T Lコンパイラ（独自仕様言語コンパイラ 3 0 に相当する。）の呼び出しによって実行され、図 1 2 のステップS 4 0 2と同様に、上記したCオブジェクトコードで表わされる機械語（ある特定のテストプロセッサで理解できる機械語）とは異なる旧テストプロセッサ仕様の機械語への翻訳を行う。

#### 【 0 0 7 4 】

A T Lオブジェクトコードの生成が終わると、統合コンパイラ 3 4 は、ステップS 1 2 2において生成されたCオブジェクトコードに、ステップS 1 2 1において生成されたA T Lオブジェクトコードを結合し、さらにそのA T Lオブジェクトコードが格納された位置情報（A T Lオブジェクトコード開始位置）を付加したオブジェクトファイル（以下、C + A T Lオブジェクトファイルと称する。）を生成する（ステップS 1 2 4）。図 6 は、このC + A T Lオブジェクトファイルの構成を示す図である。図 6 に示すように、C + A T Lオブジェクトファイルは、Cオブジェクトコードに続いてA T Lオブジェクトコードが配置される。なお、同図において、A T Lオブジェクトコードの位置情報等の付加情報の図示は省略されている。

#### 【 0 0 7 5 】

なお、この統合コンパイラ 3 4 によるコンパイル処理は、上記同様に作成され

た複数のC+ATLソースファイルに対して行われ、これにより複数のC+ATLオブジェクトファイルが用意される。このようにして統合コンパイラ34によるコンパイル処理が終わると、プログラム開発者は、リンカ33に対し、生成された複数のC+ATLオブジェクトファイルやその他に必要なライブラリファイルを指定してリンクを実行させる（図2のステップS130）。

#### 【0076】

リンカ33は、複数のC+ATLオブジェクトファイルやその他に必要なライブラリファイルに加え、上記各C+ATLオブジェクトファイルからATLオブジェクトコード部分をロードするためのロードプログラムを用意し、それらをリンクすることによって、半導体試験装置11のテストプロセッサ上で直接実行可能な単一オブジェクトファイルを生成する。

#### 【0077】

このような手順によって、単一オブジェクトファイルが用意されると、プログラム開発者は、ワークステーション10上において、半導体試験装置11との通信を可能にするコントロールプログラムを起動し、そのコントロールプログラムを用いて上記単一オブジェクトファイルを半導体試験装置11のテストプロセッサへと転送する（ステップS140）。

#### 【0078】

続いて、プログラム開発者は、上記したコントロールプログラムに対して、単一オブジェクトファイルの実行指示を与える（ステップS150）。これにより、半導体試験装置11のテストプロセッサは、まず、単一オブジェクトファイルに含まれているロードプログラムに従って、同単一オブジェクトファイル内に配置されているCオブジェクトコードとATLオブジェクトコードをメモリ上にロードする。続いて、テストプロセッサは、ロードされたCオブジェクトコードに記述されたアルゴリズムに従って、同じくロードされているATLオブジェクトコードの実行→テスト本体の所望の試験ユニットの稼動→被測定デバイスから得られた試験結果の取得→データ処理を繰り返す。この際、データ処理によって適宜加工等が行われた試験結果は、従来と同様に、半導体試験装置11の通信インターフェース、通信ケーブル、ワークステーション10の通信インターフェース

41を介して、上記したコントロールプログラムで受け取ることができ、そのコントロールプログラムに割り当てられたウィンドウ画面上に表示される。

#### 【0079】

なお、ここでは、単一オブジェクトファイル内に、ロードプログラムが含まれているとしたが、半導体試験装置11のテストプロセッサ内において、あらかじめそのロードプログラムを読み込ませておき、ワークステーション10からの実行指示に従って、まずこのロードプログラムが起動するようにすることもできる。

#### 【0080】

次に、本実施の形態における半導体試験システムのデバック処理について説明する。プログラム開発者は、上記したステップS150の実行によって得た試験結果が、明らかに異常である場合等の不具合を発見した場合、従来と同様、デバイステストプログラムに対してデバック処理を行う。そこでまず、プログラム開発者は、ワークステーション10上において、統合デバッガ35を起動させ、C＋ATLソースファイル中の所定のステートメントにブレークポイントを設定する。

#### 【0081】

そして、プログラム開発者がデバック開始を指示することによって、統合デバッガ35は、再度、上記したステップS120～S150の手順によって単一オブジェクトファイルを実行し、実行されたステートメントが、設定したブレークポイントに達したことを検出すると、Cデバッガ（汎用言語デバッガ31に相当する。）とATLデバッガ（独自仕様言語デバッガ32に相当する。）のいずれかのデバッガを起動させるかのデバッガ選択ルーチンを実行する。

#### 【0082】

図7は、このデバッガ選択ルーチンを示すフローチャートである。統合デバッガ35は、単一オブジェクトファイル中において順次実行されているステートメントがブレークポイントに到達すると、そのブレークポイントが設定されたステートメントを表示する（ステップS201）。そして、そのステートメントがATLオブジェクトコード部に相当するならば（ステップS202：Yes）、A

TL デバッガを起動し（ステップ S 206）、ATL デバッガからブレイクしたステートメントに含まれる変数等のデバッグ情報を取得する（ステップ S 207）。なお、ATL デバッガは、上記したブレイクポイント設定時に、統合デバッガ 35 によって ATL オブジェクトコード部のブレイクポイント設定情報を取得している。

#### 【0083】

統合デバッガ 35 は、ATL デバッガからデバッグ情報を取得すると、ブレイク時に有効となっている指定変数（シンボル）を表示する（ステップ S 205）。図 8 は、統合デバッガ 35 の実行画面の例を示す図であり、特に ATL 記述部内でブレイクした状態を示している。図 8 において、統合デバッガ 35 は、実行ウィンドウ 50 内に、ウィンドウ表示に標準として付加されるタイトルバーやメニューバーに加え、ブレイクポイント設定領域 51、ソース表示領域 52、シンボル表示領域 53 を有している。ここでは、C+ATL ソースプログラム中の ATL 記述部内でブレイクした状態として、ブレイクポイントが設定された行番号 5 のステートメントがソース表示領域 52 内に表示されるとともに、シンボル表示領域 53 内に、そのステートメントで使用されている変数と変数に格納された値が表示されている。

#### 【0084】

一方、統合デバッガ 35 は、ブレイクしたステートメントが C オブジェクトコード部に相当するならば（ステップ S 202：No）、C デバッガを起動し（ステップ S 203）、C デバッガからブレイクしたステートメントに含まれる変数等のデバッグ情報を取得する（ステップ S 204）。なお、C デバッガは、上記したブレイクポイント設定時に、統合デバッガ 35 によって C オブジェクトコード部のブレイクポイント設定情報を取得している。

#### 【0085】

統合デバッガ 35 は、C デバッガからデバッグ情報を取得すると、ブレイク時に有効となっている指定変数（シンボル）を表示する（ステップ S 205）。図 9 は、統合デバッガ 35 の実行画面の例を示す図であり、特に C 言語記述部内でブレイクした状態を示している。図 9 に示す統合デバッガ 35 の実行ウィンドウ

50は、図8と同様な構成であるが、ここでは、C＋A T Lソースプログラム中のC言語記述部内でブレイクした状態として、ブレイクポイントが設定された行番号15のステートメントがソース表示領域52内に表示されるとともに、シンボル表示領域53内に、そのステートメントで使用されている変数と変数に格納された値が表示されている。

#### 【0086】

プログラム開発者は、この統合デバッガ35のウィンドウ画面上に表示された変数の値を確認することによって論理的なエラーを発見した場合、エディタ28を起動させ、C＋A T Lソースファイルを適宜修正して上記したステップS120～S150の手順を繰り返す。

#### 【0087】

以上に説明したとおり、実施の形態にかかる半導体試験システム、すなわち本発明にかかるプログラム開発支援装置、プログラム実行装置、コンパイル方法およびデバッグ方法によれば、独自仕様言語プログラムであるA T Lソース自体を汎用言語プログラムであるC言語ソース内に埋め込むことによって、従来別管理されていた双方のソースを一つのC＋A T Lソースファイルとして取り扱うことができるので、ファイル管理を楽にし、プログラムの開発効率を向上させることができる。

#### 【0088】

なお、本実施の形態では、本発明にかかるプログラム開発支援装置とプログラム実行装置をそれぞれワークステーションと半導体試験装置に適用した場合を例示したが、プログラム開発支援装置を汎用的なコンピュータシステムとし、プログラム実行装置をそのコンピュータシステムと通信可能な測定装置や制御装置に適用することができるのは言うまでもない。

#### 【0089】

##### 【発明の効果】

以上に説明したように、本発明にかかるプログラム開発支援装置、プログラム実行装置、コンパイル方法およびデバッグ方法によれば、独自仕様言語プログラム自体を汎用言語プログラム内に埋め込むことで、従来別管理されていた双方の

ソースプログラムを、ソースファイルのみでなく、コンパイルによって作成されるオブジェクトファイルの段階においても一つのファイルとして取り扱うことができるので、ファイル管理を楽にし、プログラムの開発効率を向上させることができるという効果を奏する。

【図面の簡単な説明】

【図 1】

実施の形態にかかる半導体試験システムの概略構成を示すブロック図である。

【図 2】

デバイステストプログラムの開発と実行手順を示したフローチャートである。

【図 3】

C + A T L ソースプログラムの記述例を示す図である。

【図 4】

統合コンパイラによるコンパイル処理を説明するためのフローチャートである。

【図 5】

A T L ソースファイルの生成処理を説明するための説明図である。

【図 6】

C + A T L オブジェクトファイルの構成を示す図である。

【図 7】

デバッガ選択ルーチンを示すフローチャートである。

【図 8】

A T L 記述部内でブレークした状態の統合デバッガの実行画面の例を示す図である。

【図 9】

C 言語記述部内でブレークした状態の統合デバッガの実行画面の例を示す図である。

【図 10】

従来の半導体試験システムの概略構成を示すブロック図である。

【図 11】

従来の半導体試験システムのワークステーションの概略構成を示すブロック図である。

【図 12】

従来のデバイステストプログラムの開発と実行手順を示したフローチャートである。

【符号の説明】

- 10, 200    ワークステーション
- 11, 100    半導体試験装置
- 20, 220    プロセッサ
- 21, 111, 221    OSカーネル
- 22, 222    GUI処理部
- 23, 112, 223    通信バスドライバ
- 24, 224    ハードディスクドライバ
- 25, 225    マウスドライバ
- 26, 226    キーボードドライバ
- 27, 227    ディスプレイドライバ
- 28, 228    エディタ
- 29, 229    汎用言語コンパイラ
- 30, 230    独自仕様言語コンパイラ
- 31, 231    汎用言語デバッガ
- 32, 232    独自仕様言語デバッガ
- 33, 233    リンカ
- 34    統合コンパイラ
- 35    統合デバッガ
- 41, 140, 241    通信インターフェース
- 42, 242    ハードディスク装置
- 43, 243    マウス
- 44, 244    キーボード
- 45, 245    ディスプレイ装置

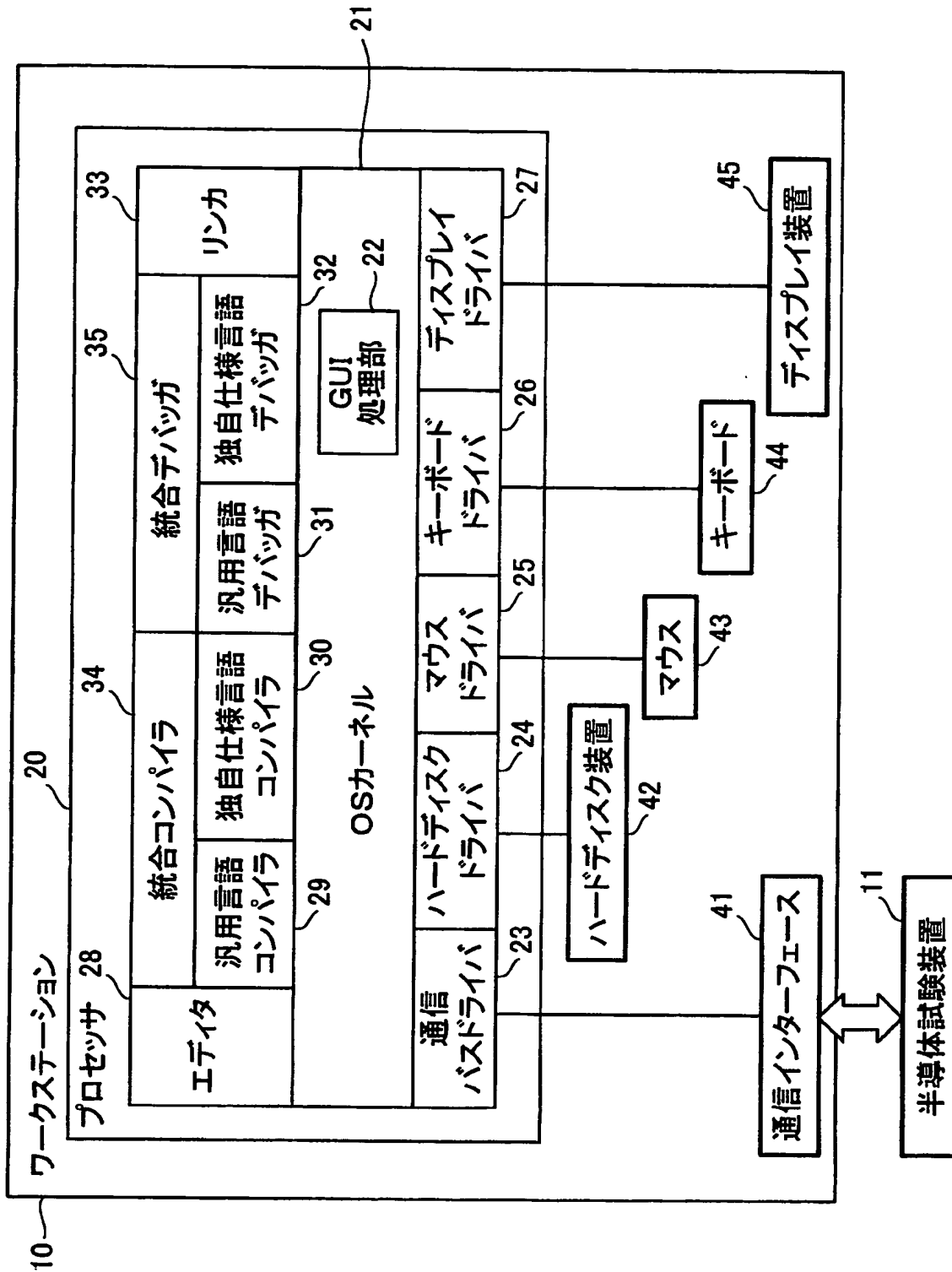
- 5 0 実行ウィンドウ
- 5 1 ブレークポイント設定領域
- 5 2 ソース表示領域
- 5 3 シンボル表示領域
- 1 1 0 テスタプロセッサ
- 1 1 3 テスタバスドライバ
- 1 1 4 汎用言語プログラム
- 1 1 5 実行用エミュレータ
- 1 1 6 制御用エミュレータ
- 1 1 7 独自仕様言語プログラム
- 1 2 0 テスタ本体
- 1 2 1 レジスタ
- 1 2 2 メモリ
- 1 2 3 試験信号送受信部
- 1 3 0 テストヘッド
- 1 3 1 被測定デバイス



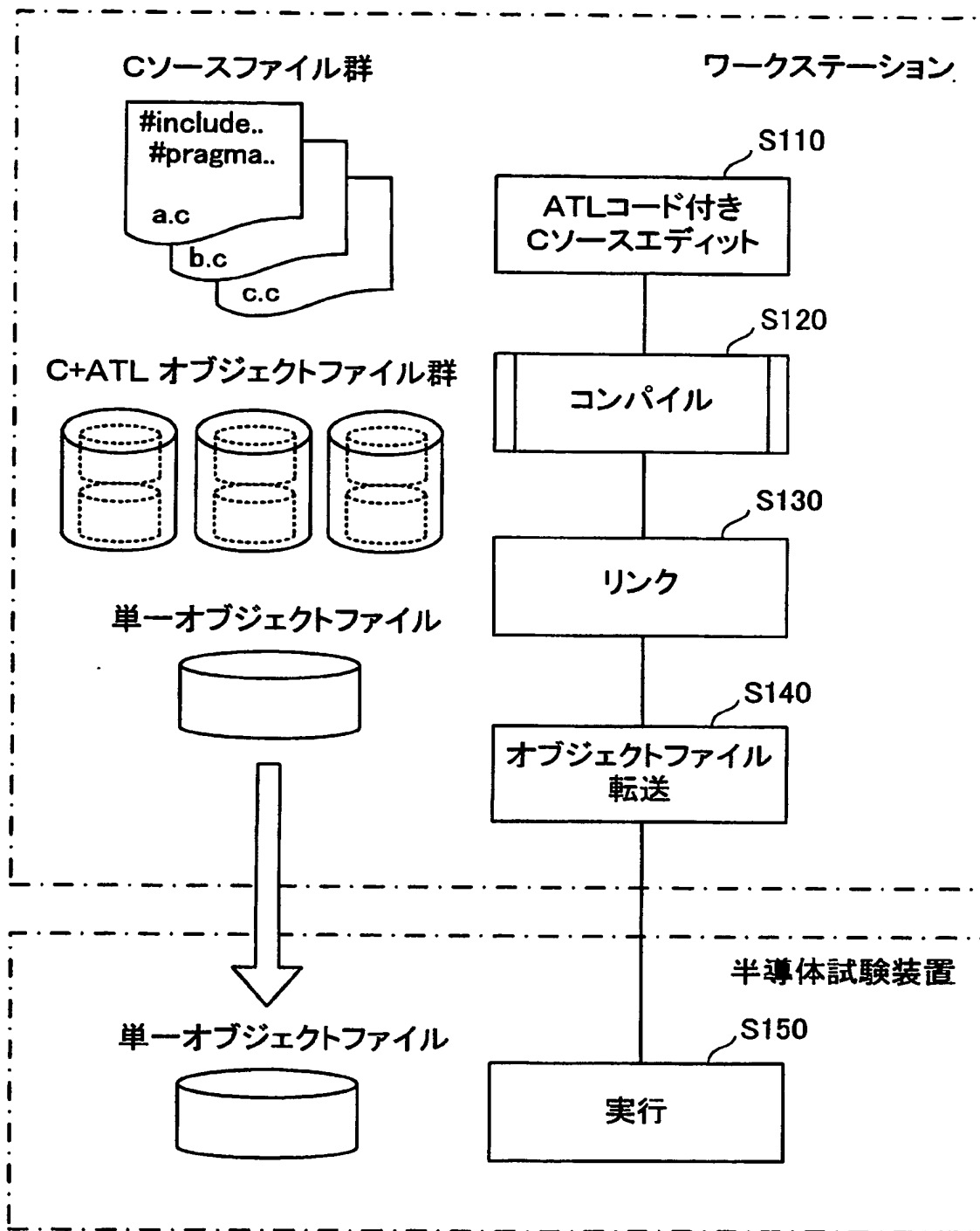
【書類名】

図面

【図 1】



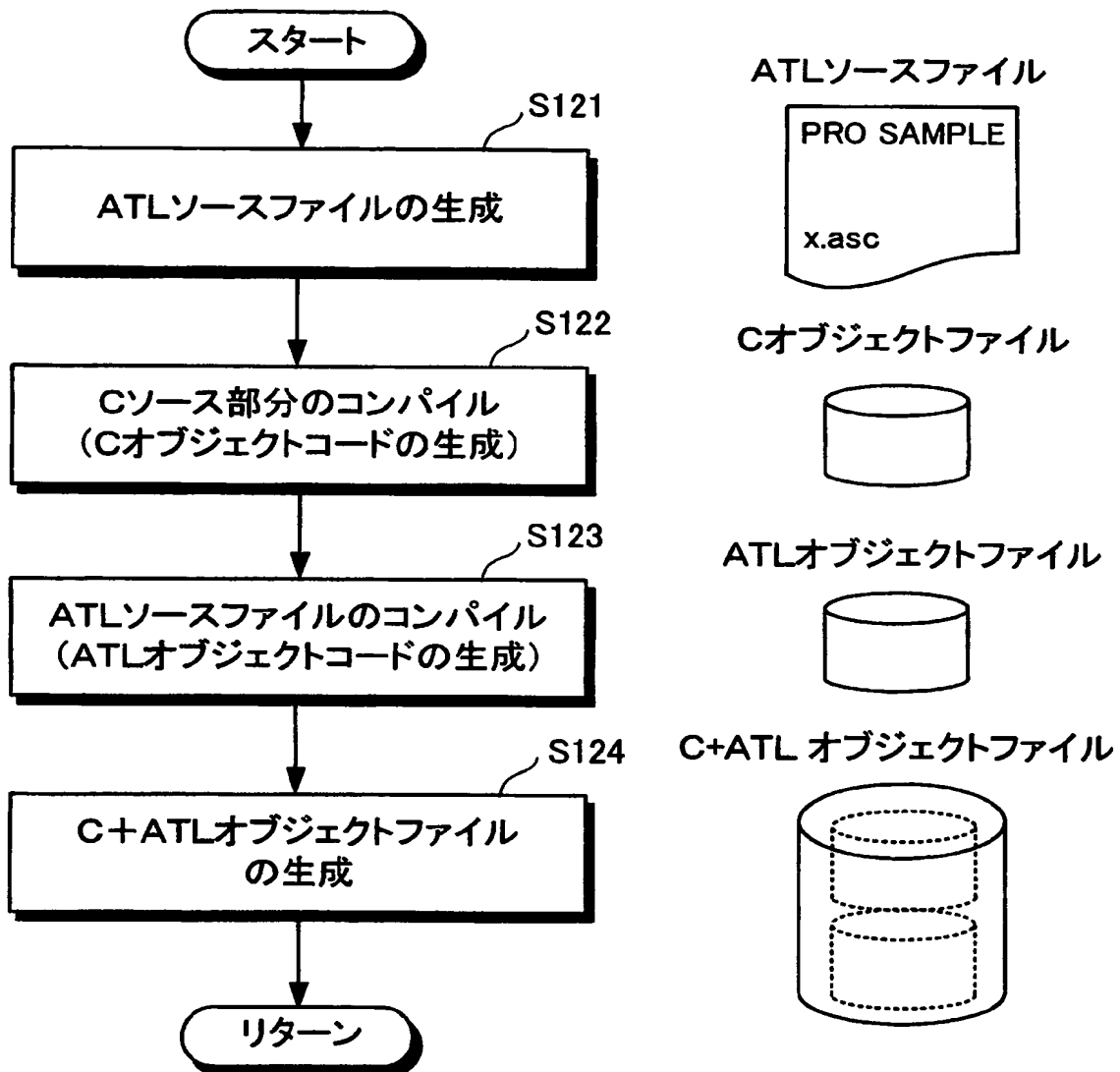
【図 2】



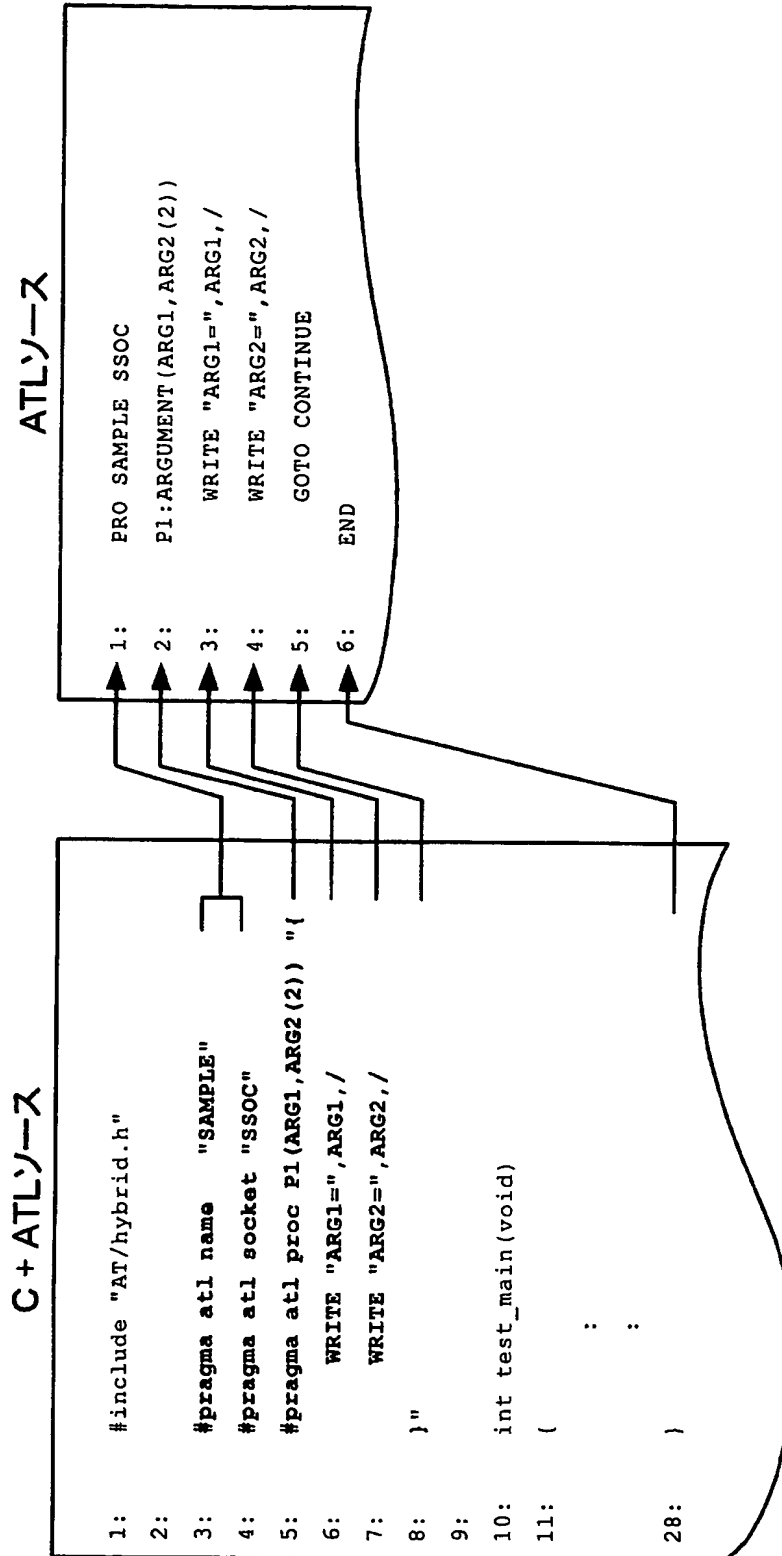
【図 3】

```
1:      #include "AT/hybrid.h"
2:
3:      #pragma atl name      "SAMPLE"
4:      #pragma atl socket    "SSOC"
5:      #pragma atl proc P1(ARG1,ARG2(2)) "{
6:          WRITE "ARG1=",ARG1,/
7:          WRITE "ARG2=",ARG2,/
8:      }"
9:
10:     int test_main(void)
11:     {
12:         int idata;
13:         struct atreal rdata[2];
14:
15:         if(ATL_init2(EM_ATL) != 0){
16:             AT_perror("ATL_init2()");
17:             exit(1);
18:         }
19:
20:         idata = 1;
21:         AT_strtorm(rdata, 2, "0.5V", "1.2V");
22:         if(ATL_proc(EM_ATL, "P1", INTE(&idata), DREAL(rdata, 2)) != 0){
23:             AT_perror("P2");
24:             exit(1);
25:         }
26:
27:         return(0);
28:     }
```

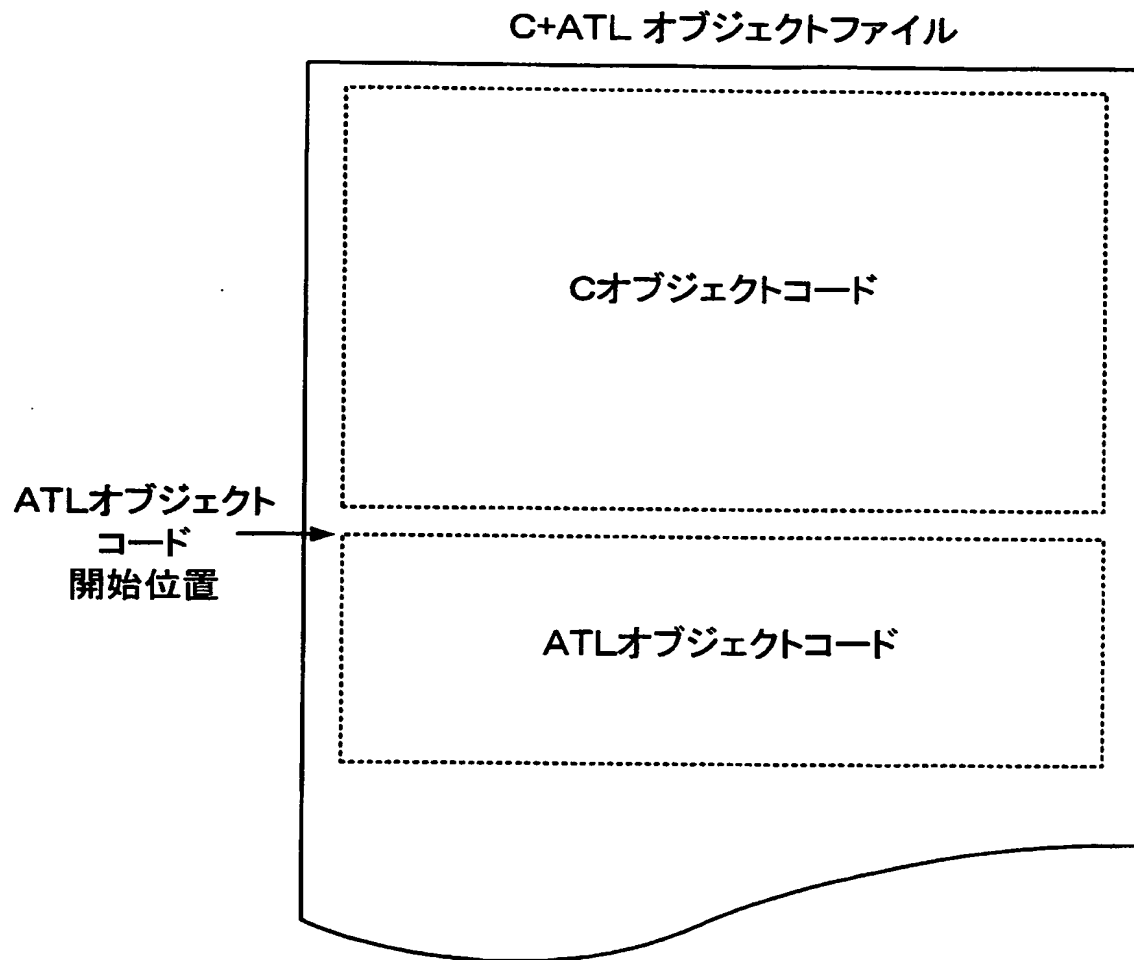
【図4】



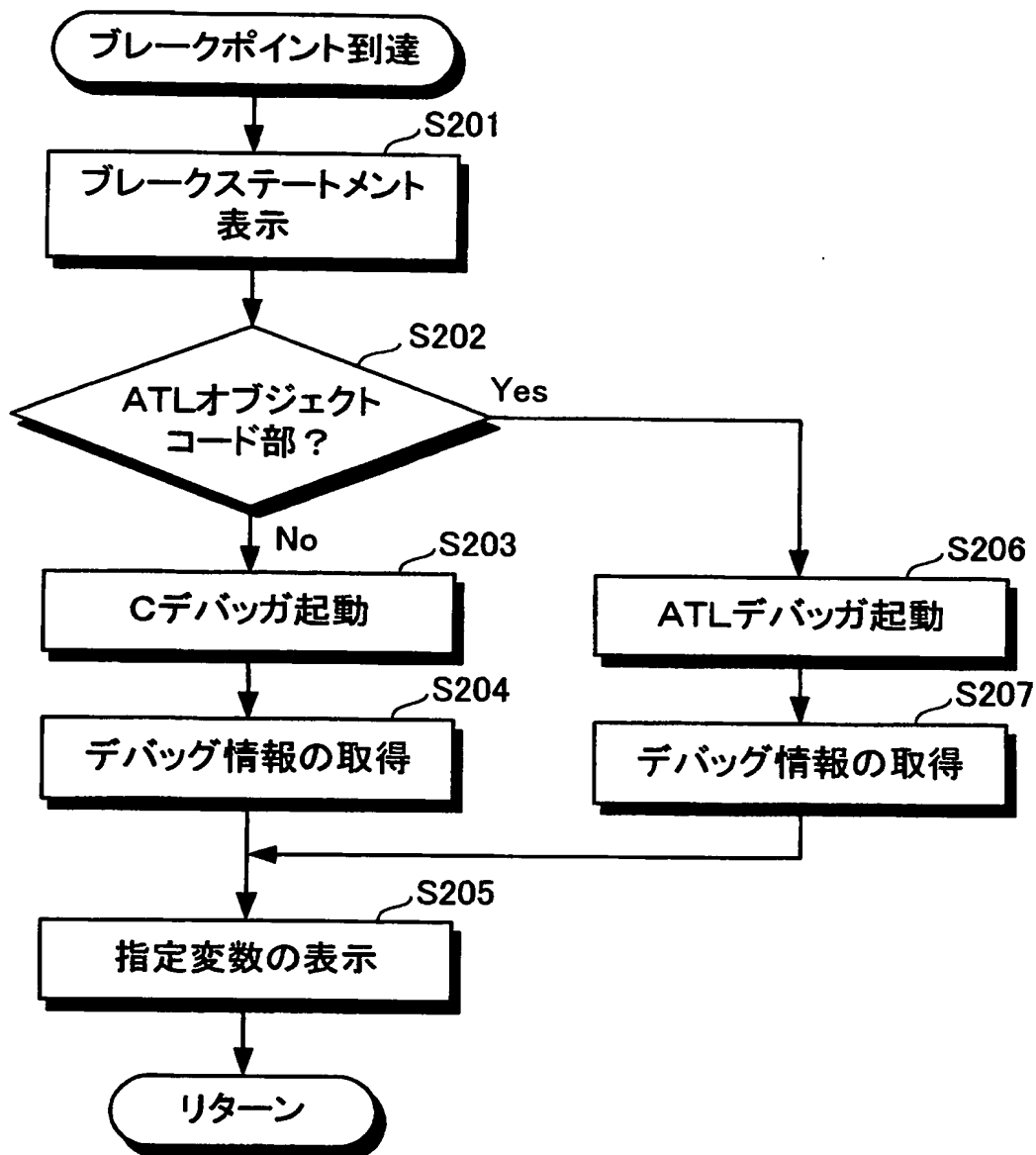
【図 5】



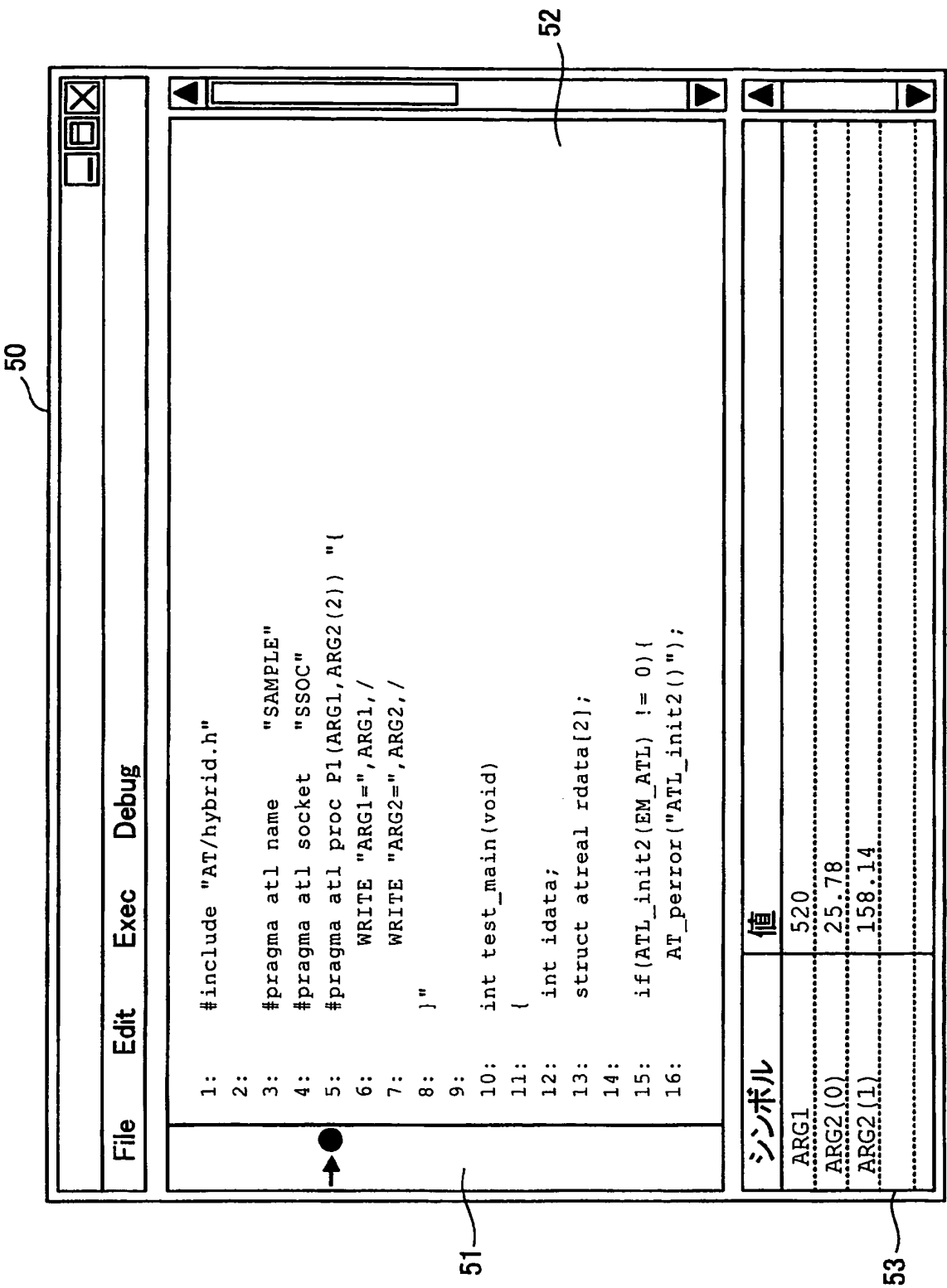
【図 6】



【図 7】

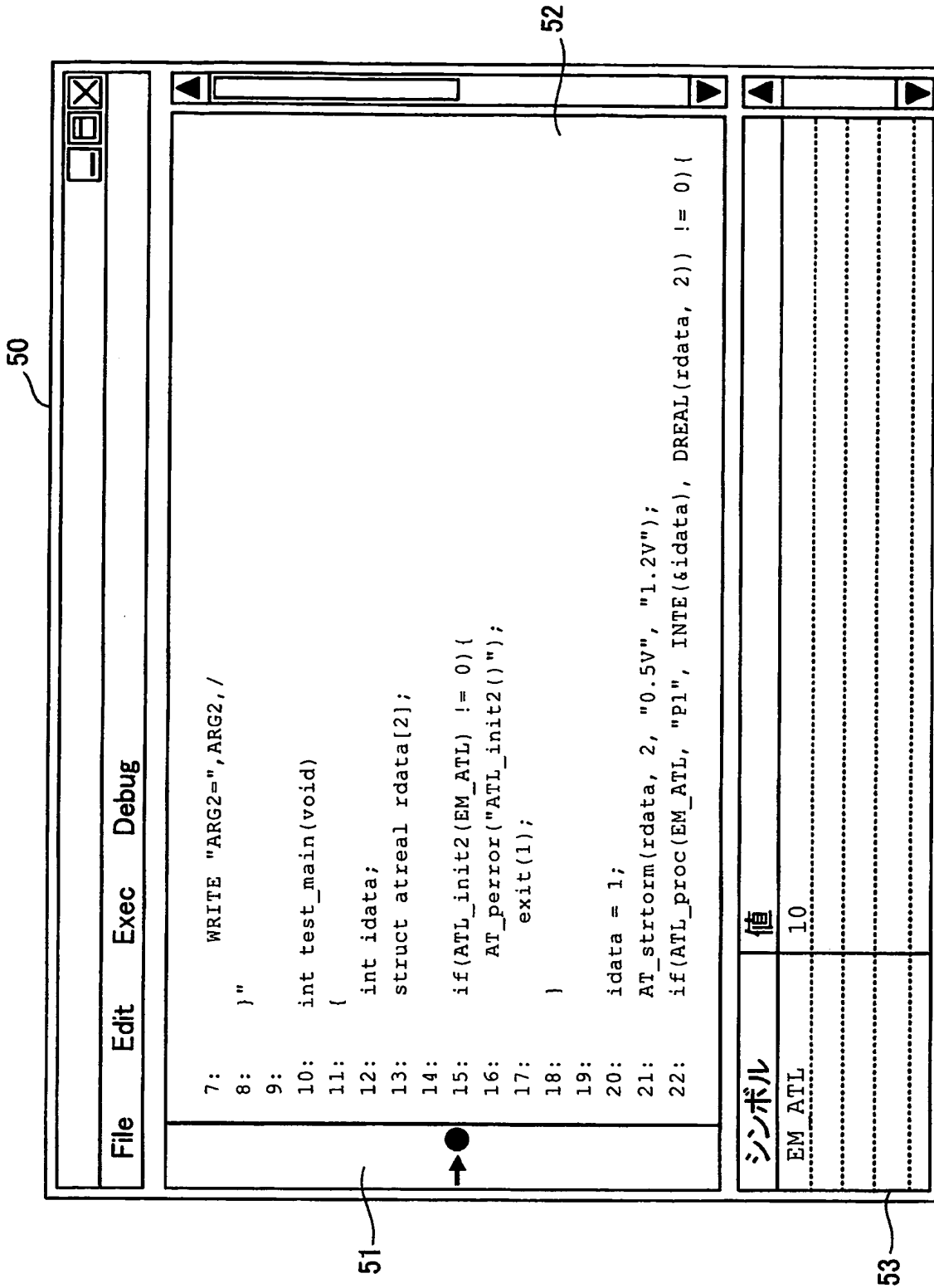


【図 8】

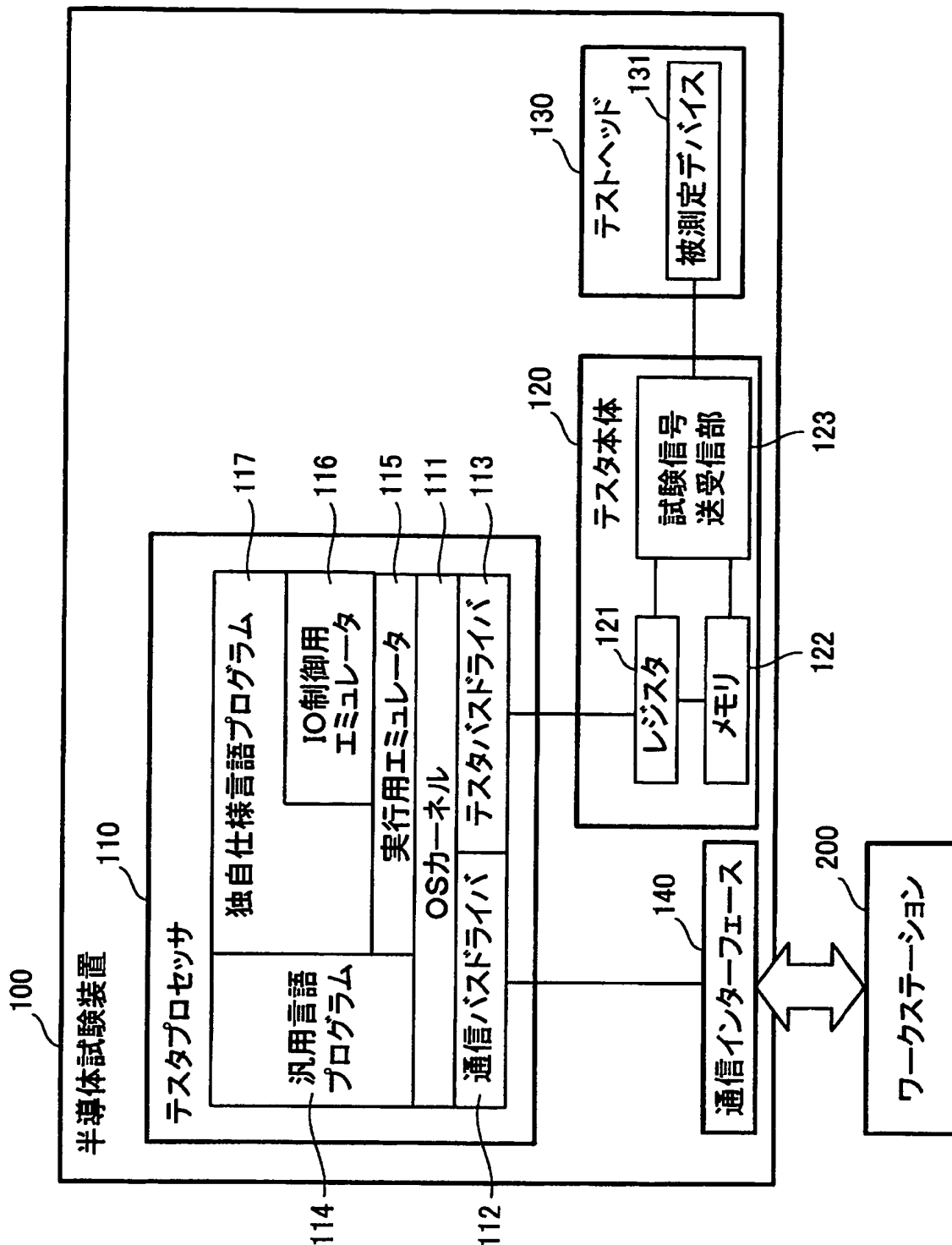




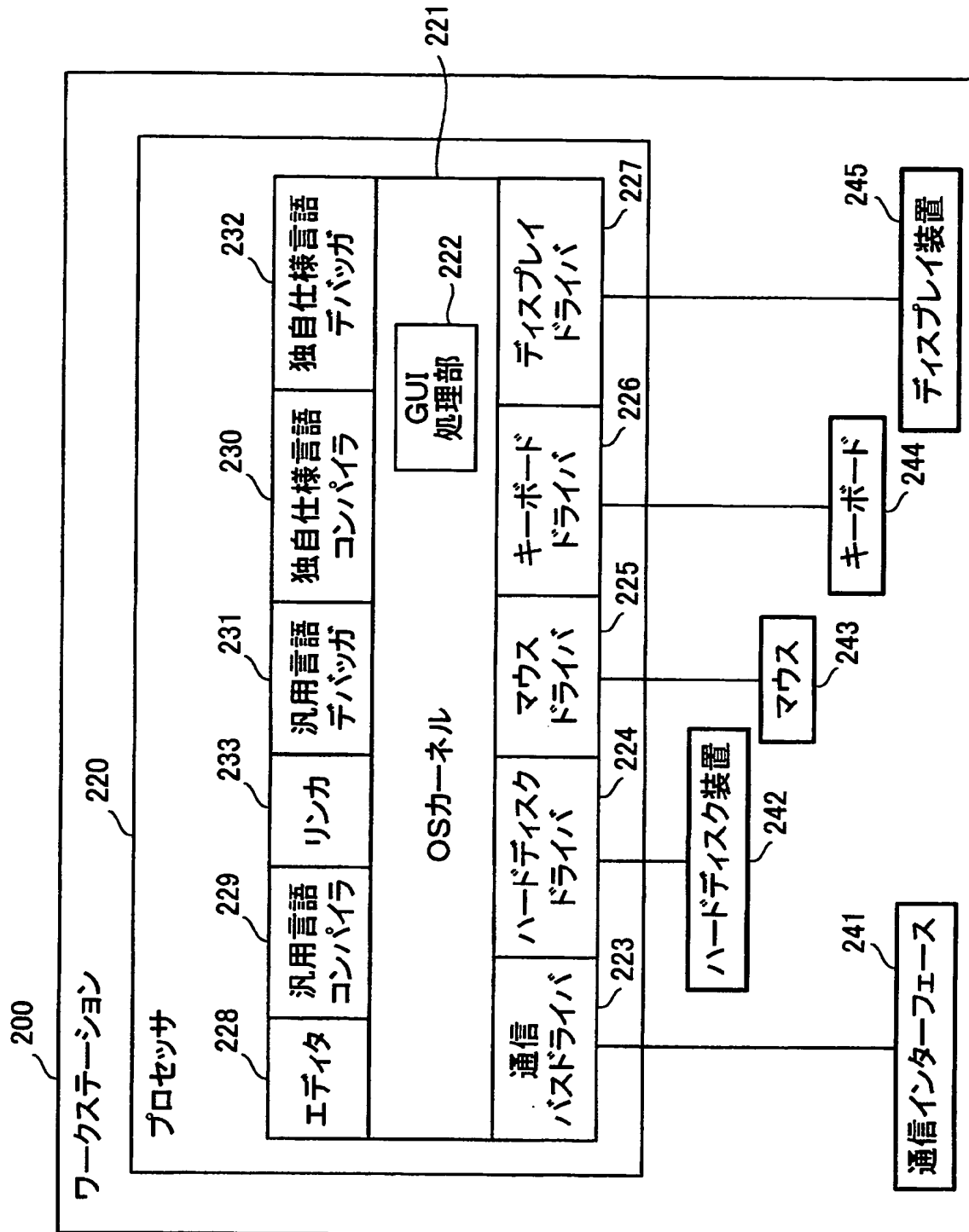
【図 9】



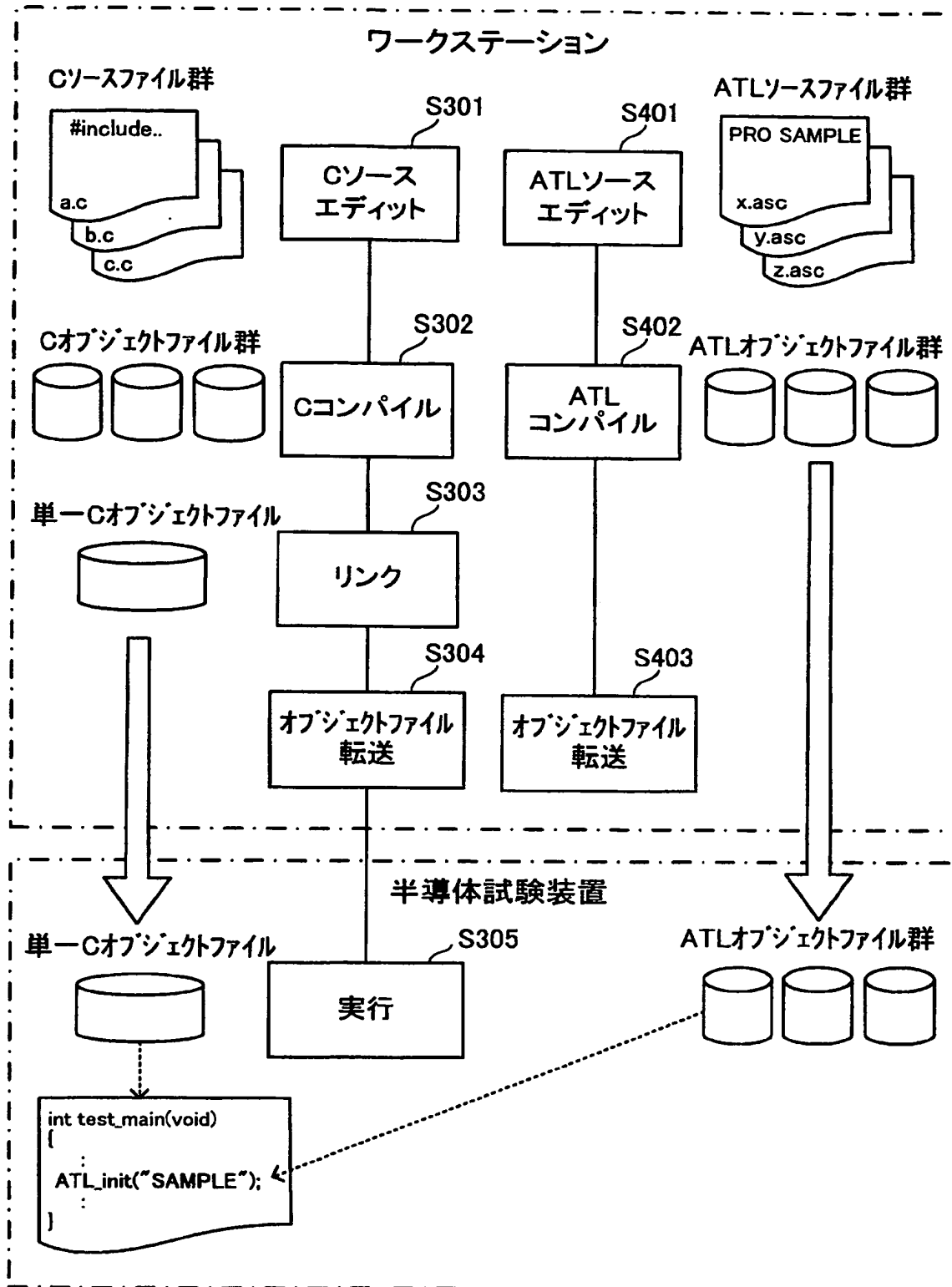
【図 10】



【図 11】



【図 12】



【書類名】 要約書

【要約】

【課題】 異種言語のプログラムが実行される環境において、開発時および実行時に必要なプログラムファイルの数を減少させることのできるプログラム開発支援装置、プログラム実行装置、コンパイル方法およびデバッグ方法を提供する。

【解決手段】 ワークステーション（プログラム開発装置に相当）上で、汎用言語ソースプログラム（図ではC言語）のプリプロセッサ記述部に、独自仕様言語ソースプログラム（図ではATL）の記述を埋め込んだ異種言語混在ソースプログラムを作成する（ステップS110）。そして、その異種言語混在ソースプログラムから汎用言語ソースプログラム記述部と独自仕様言語ソースプログラム記述部を取り出して、それぞれのコンパイラでコンパイルし、得られたそれぞれのオブジェクトコードを結合して、一つのオブジェクトファイルを生成する（ステップS120）。

【選択図】 図2

特願 2 0 0 2 - 3 0 5 0 1 0

出 願 人 履 歴 情 報

識別番号

[ 3 9 0 0 0 5 1 7 5 ]

1. 変更年月日

1 9 9 0 年 1 0 月 1 5 日

[変更理由]

新規登録

住 所

東京都練馬区旭町 1 丁目 3 2 番 1 号

氏 名

株式会社アドバンテスト